DB Qual 2011

This exam consists of two parts. Please answer each part in a separate blue booklet. You have a total of 60 minutes for the entire exam. You can allocate time among the parts as you like, but be sure not to get stuck in one part without spending some time in the other parts.

The exam is open book and notes. Laptops are only allowed for viewing notes. Simple calculators are allowed.

If you feel that a question is ambiguous or unclear, state any reasonable assumptions you need to make in order to answer the question.

This exam may be a bit on the long side. Answer as many questions as you can in the allowed time. Start with the questions you can answer quickly, and only then move to the ones that may take you more time.

DB Qual 2011; Part A

A1. (10 points) Consider the following relation with functional and multivalued dependencies:

```
Likes(name,age,food,drink)
```
$\texttt{name} \rightarrow \texttt{age}$
$\texttt{food} \rightarrow \texttt{drink}$
$\texttt{name}, \texttt{age} \twoheadrightarrow \texttt{food}$

1. **(2 points)** Give an instance of `Likes` that satisfies the dependencies. Your instance should have at least two distinct tuples (not duplicates), but otherwise as few tuples as possible.
2. **(6 points)** Decompose `Likes` into fourth normal form. If there is a unique decomposition, show the decomposition and explain briefly how you arrived at it. If there is more than one decomposition, show two of them and for each explain briefly how you arrived at it.
3. **(2 points)** Is it possible for `Hector` to like both `beer` and `wine`? A yes/no answer is sufficient. (Please base your answer on the relation and dependencies, not on any knowledge of Hector's actual (non-)drinking habits!)

A2. (5 points) Consider a relation `Menu(food,price)` where `food` is a key. Consider the following relational algebra expression over `Menu`:

$$(\Pi_{\texttt{food}}(\texttt{Menu}) - \Pi_{M1.\texttt{food}}(\sigma_{M1.\texttt{price}<M2.\texttt{price}}(\rho_{M1}(\texttt{Menu}) \times \rho_{M2}(\texttt{Menu})))) \ \cup$$
$$(\Pi_{\texttt{food}}(\texttt{Menu}) - \Pi_{M1.\texttt{food}}(\sigma_{M1.\texttt{price}>M2.\texttt{price}}(\rho_{M1}(\texttt{Menu}) \times \rho_{M2}(\texttt{Menu}))))$$

1. **(3 points)** State in English what this expression returns.
2. **(2 points)** Suppose relation `Menu` contains $m > 2$ distinct tuples but otherwise we know nothing of its contents. What are the minimum and maximum number of tuples that could be returned by the expression?

A3. (6 points) Consider a hybrid of the previous two relations: `Eats(name,food,price)`. Do not make any assumptions about keys or dependencies, but you may assume there are no duplicates (i.e., all three attributes together form a key). Write a SQL query to find the average of the most expensive foods eaten by each person. (For example, if `Jennifer` eats foods with prices 8 and 10, `Hector` eats foods with prices 50, 55, and 60, and `Marianne` eats foods all priced at 20, then the answer is 30.) *Your solution will be graded on simplicity as well as correctness.*

(problems continue on next page)

A4. (4 points) Consider the same relation `Eats(name,food,price)` from the previous problem, and the following transaction `T1` over `Eats`:

```
T1: Select Avg(price) From Eats;
    Select Sum(price) From Eats;
    Commit;
```

Specify another transaction `T2` such that the results of transaction `T1` may be different in the following two scenarios:

- Both transactions are executed with isolation level `Serializable`.
- Your transaction `T2` is executed with isolation level `Serializable`, but transaction `T1` is executed with isolation level `Repeatable-Read`.

Your solution will be graded on simplicity as well as correctness.

A5. (5 points) Consider the following DTD for XML documents:

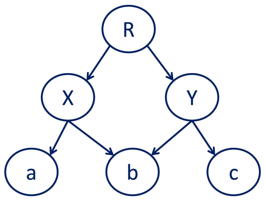
```
<!DOCTYPE Foods [
   <!ELEMENT Foods (Person*)>
   <!ELEMENT Person (Name, Likes*)>
   <!ELEMENT Name (#PCDATA)>
   <!ELEMENT Likes EMPTY>
   <!ATTLIST Likes Food CDATA #REQUIRED> ]>
```

Write an expression in XPath that operates on a document "foods.xml" conforming to this DTD. The XPath expression should return the names of all people who like `pizza` but don't like `salad`. *Your expression will be graded on simplicity as well as correctness.*

DB Qual 2011; Part B

Problem B1 (10 points)

Consider a multi-granularity, two phase locking scheme with the following types of locks: IS, IX, S, X, SIX. We want to extend the traditional scheme to work on directed acyclic graphs (DAGs); for this problem let us focus exclusively on the following DAG:



In this DAG, note that object b can be reached via two paths, $R \rightarrow X \rightarrow b$ and $R \rightarrow Y \rightarrow b$. We want an *efficient* locking scheme that minimizes the number of locks a transaction needs to request. Let us assume that reads are more frequent than writes.

Illustrate how locking should work by stating the locks (in order) requested by the following transactions. Each transaction performs only the listed actions, nothing else. (Note that each case below is independent of the others.)

(a) T_1 reads object a.

(b) T_2 reads objects a and b.

(c) T_3 writes objects a and b.

(d) T_4 reads all objects in Y and writes only object b.

Problem B2 (10 points)

(a) Consider the following three transactions:

- S: $[X := X + 2]$,
- T: $[X := X + 3; Y := Y - 3]$, and
- U: $[Y := Y + 4; Z := Z - 4]$.

Assume that a database system using undo logging crashes with the log records on disk given below:

- (START S);
- (S, X, 10);
- (START T);
- (COMMIT S);
- (T, X, 12);
- (T, Y, 25);
- (START U);
- (COMMIT T);
- (U, Y, 22);
- (U, Z, 10)

Determine all possible values of X, Y and Z that could appear on disk at the time of the crash.

Possible X values:____________________________

Possible Y values:____________________________

Possible Z values:____________________________

(b) Let a database contain initial values of $X = 10$, $Y = 20$ and $Z = 5$ on disk. Assume a redo logging scheme is in use, and let the redo log contain the sequence of records (the transactions are different from those in part (a)):

- (START, S);
- (S, X, 15);
- (S, Y, 22);
- (COMMIT S);
- (START T);
- (T, X, 18);
- (START U);
- (U, Y, 26);
- (T, Z, 25);
- (COMMIT U).

Determine all possible values of X, Y and Z that could appear on disk at the time of the crash.

Possible X values:____________________________

Possible Y values:____________________________

Possible Z values:____________________________

Problem B3 (10 points)

You are designing a B+ tree index for your database system, and you are considering the following two designs:

- *Conventional:* All B+ tree nodes can have a maximum of n keys. Leaf nodes also contain at most n keys. The pointers associated with the leaf keys point to the actual records, that are stored elsewhere. Thus, to lookup a record given its key (assuming key exists), you need $d + 1$ IOs, where d is the depth of the tree.
- *Compact:* The full records are stored in the leaves of the tree, thus leaf nodes can at most hold n' keys. We assume the full record is larger than a pointer to the records, hence $n' < n$. Non-leaf nodes hold at most n keys, as before. In this case, to find a record given its key (assuming key exists) will take d IOs, where d is the depth of the tree.

(a) Assume we want to take at most k IOs to fetch a record given its key, for the case the record is indexed. Assume there are no duplicate keys. What is the *maximum* number of records we can store in a conventional B+ tree, to meet this bound? Your answer should be a function of n, n' and k (not d).

(b) Under the same scenario as part (a), what is the *maximum* number of records we can store in a compact B+ tree, to meet this bound? Your answer should be a function of n, n' and k (not d).

(c) What type of index should you implement, assuming you have such a bound on the number of IOs for a successful find?

(d) Assume we want to take at most k IOs to discover that a key does not exist in the index. Assume there are no duplicate keys. What is the *maximum* number of records we can store in a conventional B+ tree, to meet this different bound? Your answer should be a function of n, n' and k (not d).

(e) Under the same scenario as part (d), what is the *maximum* number of records we can store in a compact B+ tree, to meet this bound? Your answer should be a function of n, n' and k (not d).

(f) What type of index should you implement, assuming you have a bound on the number of IOs for an *unsuccessful* find?